BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

As rescanning documents *will not* correct images, please do not report the images to the Image Problem Mailbox.

L Number	Hits	Search Text	DB	Time stamp
1	20	(linear adj data) with (web stream)	USPAT	2003/07/18 10:27
2	38	(linear adj data) and (709/$ 712/$).ccls.	USPAT	2003/07/18 10:31
3	1	6185733.pn. and (internet network$3)	USPAT	2003/07/18 10:34
4	0	6185733.pn. and (open)	USPAT	2003/07/18 10:35
5	17	open adj file adj operation$3	USPAT	2003/07/18 10:34
6	1	6185733.pn. and (input output)	USPAT	2003/07/18 10:35
7	1	6185733.pn. and (output)	USPAT	2003/07/18 10:37
8	1	6185733.pn. and (view display$5)	USPAT	2003/07/18 10:39
9	1	(read adj file adj operation$3) with (input output i/o)	USPAT	2003/07/18 10:38
10	1	6185733.pn. and (read$5)	USPAT	2003/07/18 10:42
11	0	6185733.pn. and (server$5 with (memory stor$5 buffer$5))	USPAT	2003/07/18 10:56
12	496	library with directory	USPAT	2003/07/18 10:51
13	0	6185733.pn. and (cach$6)	USPAT	2003/07/18 11:40
15	9020	(determin$5 adj9 block$5) and (70?/$ 71?).ccls.	USPAT	2003/07/18 11:43
-	1	(palevich near john near h).in. (taillefer near martin).in.	USPAT	2003/07/15 11:28
-	2	((set near top near box) (set-top near box)) with (file near system$3)	USPAT	2003/07/15 16:27
-	0	((set near top near box) (set-top near box)) with (window near ce)	USPAT	2003/07/15 15:53
-	0	(file near system$5) with (window near ce)	USPAT	2003/07/15 14:09
-	0	((set near top) (set-top)) with (window near ce)	USPAT	2003/07/16 10:39
-	18	(window near ce)	USPAT	2003/07/15 14:10
-	46	(file near system$3) with (compress$5 near (file$5 format$5))	USPAT	2003/07/15 14:12
-	1	(file near system$3) with (compress$5 near (file$5 format$5)) same http	USPAT	2003/07/15 14:13
-	2	(("6408403") or ("6349393")).PN.	USPAT	2003/07/15 14:27
-	888	(convert$6 convers$6) with (block$5) with (stream$6)	USPAT	2003/07/15 17:15
-	76	((convert$6 convers$6) with (block$5) with (stream$6)) and (compress$6 near (format$5 file$5))	USPAT	2003/07/15 14:33
-	48	((convert$6 convers$6) with (block$5) with (stream$6)) and ((set adj top) set-top)	USPAT	2003/07/15 14:33
-	0	((set near top) (set-top)) with (block adj driver$5)	USPAT	2003/07/15 15:52
-	0	(convert$6 convers$6) with (url near3 get) with (block$6)	USPAT	2003/07/15 15:54
-	5	(convert$6 convers$6) with (url (get near request$5)) with (block$6)	USPAT	2003/07/15 15:58
-	60	(convert$6 convers$6) with (url (get)) with (block$6)	USPAT	2003/07/15 15:57
-	19	(convert$6 convers$6) with (url http (get near request$5)) with (block$6)	USPAT	2003/07/15 16:27
-	382	windows near CE	USPAT	2003/07/15 16:42
-	28	((set near top near box) (set-top near box)) and (windows near CE)	USPAT	2003/07/15 16:42
-	2	(ATDDISK ATAPI (secure near digital near card) (block adj driver$5)) and (windows near CE)	USPAT	2003/07/15 16:58
-	22	(ATDDISK ATAPI (secure near digital near card) (block adj driver$5)) and (set-top (set adj top))	USPAT	2003/07/16 16:15
-	5	(convert$6 convers$6) with (read adj request$6) with (access$5 near request$6)	USPAT	2003/07/15 16:54
-	0	(block adj3 driver$5) same (access adj file$6) same (read adj file$5)	USPAT	2003/07/15 16:59
-	19	(block adj3 driver$5) with (access$5 and read$6)	USPAT	2003/07/15 17:00
-	0	(convert$6 convers$6) with (read adj request$6) with (http url)	USPAT	2003/07/15 17:03
-	15	(read adj request$6) with (http url)	USPAT	2003/07/15 17:10
-	230	(read adj request$6) with (access adj request$3)	USPAT	2003/07/15 17:10
-	0	(convert$6 convers$6) with ((http ftp) near9 (get and put))	USPAT	2003/07/15 17:16

-	17	((http ftp) near9 (get and put))	USPAT	2003/07/15 17:33
-	8	http with byte with range$5	USPAT	2003/07/15 17:35
-	0	6314456.PN. AND (http adj '1.1')	USPAT	2003/07/15 17:35
-	0	6314456. AND (http)	USPAT	2003/07/15 17:35
-	0	6314456. and (http)	USPAT	2003/07/15 17:36
-	0	6314456.pn. and (http adj '1.1')	USPAT	2003/07/15 17:36
-	1	6314456.pn. and (http)	USPAT	2003/07/15 17:43
-	0	(access adj request$5) with (http adj3 (get post put))	USPAT	2003/07/15 17:52
-	90	(access adj request$5) with ((get post put))	USPAT	2003/07/15 17:59
-	21	(access adj request$5) with (http)	USPAT	2003/07/15 18:07
-	2	(disk near9 read near9 request$5) with (http ftp)	USPAT	2003/07/15 18:08
-	4	(((file near9 system$3) disk$6) near9 read near9 request$5) with (http ftp)	USPAT	2003/07/15 18:17
-	4	((disk (file adj2 system$5)) with ((read write) adj3 request$5)) with (http ftp telnet$5)	USPAT	2003/07/15 18:19
-	11	((disk (file adj2 system$5)) with (read write)) with (http ftp telnet$5)	USPAT	2003/07/16 10:34
-	0	((set near top) (set-top)) with (http adj server$5)	USPAT	2003/07/16 16:26
-	21	((set near top) (set-top)) with (http)	USPAT	2003/07/16 11:02
-	2	(("6138271") or ("6029000")).PN.	USPAT	2003/07/16 11:21
-	0	(compress$5 near3 file$5 near3 system$5) with http	USPAT	2003/07/16 12:16
-	5	(compress$5 near3 file$5 near3 system$5) with (block$3 near data)	USPAT	2003/07/16 11:24
-	4	(block adj data) with http	USPAT	2003/07/16 12:15
-	0	6192432.pn. and (http web url)	USPAT	2003/07/16 12:16
-	57	(compress$5 near3 file$5 near3 system$5) and (http url web)	USPAT	2003/07/16 12:31
-	1	6401239.pn. and (compress$6 decompress$5 zip$3 unzip$5)	USPAT	2003/07/16 12:51
-	19	(convert$6 convers$6) with block$6 with ((get adj request$5) http url)	USPAT	2003/07/16 12:48
-	0	compress$6 with (file$3 near3 block$6) with ((get adj request$5) http url)	USPAT	2003/07/16 12:51
-	0	6163812.pn. and (compress$6 decompress$5 zip$3 unzip$5)	USPAT	2003/07/16 12:52
-	0	6163812.pn. and (conver$6 conversion$6)	USPAT	2003/07/16 12:52
-	10	(window? adj2 ce) and ((file$5 near3 system$5) with (url http web))	USPAT	2003/07/16 13:26
-	6	file adj system adj read adj request$3	USPAT	2003/07/16 13:29
-	43	(file adj2 request$3) with ((http url web) adj3 request$5)	USPAT	2003/07/16 13:32
-	3	(convert$6 compress$6 conversion) with (file adj2 request$3) with ((http url web) adj3 request$5)	USPAT	2003/07/16 13:31
-	0	(file adj2 system? adj3 request$3) with ((http url web) adj3 request$5)	USPAT	2003/07/16 13:32
-	0	(file adj2 system? adj3 request$3) with (http url web)	USPAT	2003/07/16 13:32
-	64	(file adj2 system?) with (http url web)	USPAT	2003/07/16 13:43
-	0	(read adj request$5) with (file adj2 system?) with (http url web)	USPAT	2003/07/16 13:44
-	0	((write read) adj request$5) with (file adj2 system?) with (http url web)	USPAT	2003/07/16 14:07
-	1	6163812.pn. and (file$3 with (read$3 write request$5))	USPAT	2003/07/16 14:08
-	191	(file adj2 (read write) adj3 request$3)	USPAT	2003/07/16 14:23
-	0	(convert$6 conversion) with (os near request$5) with ((http url web) near request$6)	USPAT	2003/07/16 14:15
-	0	(os near request$5) with ((http url web) near request$6)	USPAT	2003/07/16 14:15
-	782	http adj request	USPAT	2003/07/16 14:16
-	0	6163812.uref.	USPAT	2003/07/16 14:18
-	82	(file near6 (read write) near6 request$3) and ((http url web) near9 request$5)	USPAT	2003/07/16 14:27

-	0	(convert$6 conversion$6 chang$6) with ((read write) near3 operation$3) with ((http url web) near request$5)	USPAT	2003/07/16 14:30
-	0	(convert$6 conversion$6 chang$6) with ((read write open close) near3 operation$3) with ((http url web) near request$5)	USPAT	2003/07/16 14:31
-	11	(convert$6 conversion$6 chang$6) with ((read write open close) near3 operation$3) with (http url web)	USPAT	2003/07/16 14:33
-	0	((read write open close) near3 operation$3) with ((http url web) near request$3)	USPAT	2003/07/16 14:33
-	3	((read write open close) near3 operation$3) same ((http url web) near request$3)	USPAT	2003/07/16 14:50
-	2	(("5944780") or ("5991306")).PN.	USPAT	2003/07/16 14:56
-	17	5944780.uref.	USPAT	2003/07/16 15:02
-	4	(("6115741") or ("6101558") or ("5944780") or ("5995756")).PN.	USPAT	2003/07/16 15:08
-	0	5944780.pn. and (compress$6 zip$5 unzip$6 decompress$6)	USPAT	2003/07/16 15:08
-	1	5944780.pn. and (local near cache$5)	USPAT	2003/07/16 15:40
-	1	5944780.pn. and (http with (remote network$5))	USPAT	2003/07/16 16:12
-	53	(compress$6) and (block adj driver$5)	USPAT	2003/07/16 16:13
-	106	(ATDDISK ATAPI (secure near digital near card) (block adj driver$5)) and (compress$6)	USPAT	2003/07/16 16:16
-	4	((set near top) (set-top)) and (file near driver$5) and (block$5)	USPAT	2003/07/16 16:31
-	0	(block adj driver$3) with afpa	USPAT	2003/07/16 16:34
-	68	(block$6) same ((secure adj digital adj card$3) atapi atadisk)	USPAT	2003/07/16 16:35
-	0	(block$6 same ((secure adj digital adj card$3) atapi atadisk)) and (file adj driver$5)	USPAT	2003/07/17 10:25
-	22	((read write) adj request$5) with (http url web)	USPAT	2003/07/17 10:40
-	6	compress$6 with (read write) with (http url web)	USPAT	2003/07/17 10:33
-	2	http near9 byte near9 range$5	USPAT	2003/07/17 10:34
-	2	((read write) near5 operation$5) with (http)	USPAT	2003/07/17 10:42
-	6	(read write) with (file adj system$5) with (http)	USPAT	2003/07/17 10:54
-	2	(("6161146") or ("6321334")).PN.	USPAT	2003/07/17 10:45
-	51	((read write) with http) and (file adj system$3)	USPAT	2003/07/17 10:47
-	1	(file adj4 system$5 adj4 (manager$3 controller$5)) and (web adj3 (controller$5 manager$5))	USPAT	2003/07/17 11:24
-	32	(file near system$5 near access$5) and (web near access$5)	USPAT	2003/07/17 11:13
-	43	(file near system$5 near access$5) and ((http web url) near access$5)	USPAT	2003/07/17 12:13
-	0	(file near system$5 near (manager$3 controller$5)) and (web near (controller$5 manager$5))	USPAT	2003/07/17 11:27
-	45	(convert$5 convers$6) with (format$5) with (data adj request$6)	USPAT	2003/07/17 11:33
-	0	(convert$5 convers$6) with (format$5) with ((file near system$5) and http)	USPAT	2003/07/17 11:33
-	3	(convert$5 convers$6) with (request$6) with ((file near system$5) and http)	USPAT	2003/07/17 11:42
-	5	(("6185733") or ("6163844") or ("5991798") or ("5880730") or ("5751961")).PN.	USPAT	2003/07/17 14:32
-	6	(convert$5 convers$6) with (request$6) with ((file near system$5) and (url web http))	USPAT	2003/07/17 13:05
-	0	6519626.pn. and (compress$6 decompress$6 zip)	USPAT	2003/07/17 12:11

-	2	((url http) near (subdirector$5 director$6)) with (file near (path directory))	USPAT	2003/07/17 12:13
-	159	((url http) with (subdirector$5 director$6)) with (file near4 (path directory system$5))	USPAT	2003/07/17 12:14
-	3	((url http) with (subdirector$5 director$6)) with (correspond$6 match$6) with (file near4 (path directory system$5))	USPAT	2003/07/17 12:15
-	8	(url http) with (correspond$6 match$6) with (file near4 (path directory system$5))	USPAT	2003/07/17 12:23
-	0	6519626.pn. and (reconvert$6)	USPAT	2003/07/17 12:23
-	54	file adj system adj path	USPAT	2003/07/17 13:06
-	1	(file adj system adj path) with (read write)	USPAT	2003/07/17 13:07
-	1	6519626.pn. and ((file adj system$5) with (request$5 read))	USPAT	2003/07/17 13:14
-	0	6519626.pn. and (file near3 read)	USPAT	2003/07/17 13:14
-	1596	file$3 near5 system near5 read	USPAT	2003/07/17 13:26
-	591	file$3 adj system near5 read	USPAT	2003/07/17 13:37
-	0	((read open) near9 (local near9 cache$5)) with (convert$6 convers$6) with (http url)	USPAT	2003/07/17 13:38
-	0	((read open) with (cache$5)) with (convert$6 convers$6) with (http url)	USPAT	2003/07/17 13:40
-	9	(read open) with (convert$6 convers$6) with (http url)	USPAT	2003/07/17 13:52
-	5	(file adj system$5) with (convert$6 convers$6) with (http url)	USPAT	2003/07/17 14:04
-	1	6185733.pn. and (read open request$3)	USPAT	2003/07/17 14:39
-	3	((("6185733") or ("6163844") or ("5991798") or ("5880730") or ("5751961")).PN.) and (compress$6 decompress$6 conver$6)	USPAT	2003/07/17 14:47
-	39	(decompress$6 unzip) with (file$3 data) with (url http web)	USPAT	2003/07/17 16:33
-	1	6185733.pn. and (input$3 write$5 open$3)	USPAT	2003/07/17 14:59
-	1981	(compress$6) with (reduc$6) with memory	USPAT	2003/07/17 16:49
-	285	(decompress$6) with (reduc$6) with memory	USPAT	2003/07/17 16:52
-	3	(decompress$6) with view with (original near4 (image$5 file$3))	USPAT	2003/07/18 10:23

Broadcast Services for Microsoft Windows CE Set-Top Boxes

Microsoft Corporation

June 1999

Summary: Broadcast Services (BCS) for the Microsoft **Windows CE** operating systems is a high-performance application interface that extends **Windows CE** capabilities to include broadcast audio/video services, conditional access processing, Electronic Program Guide (EPG) infrastructure, and broadcast data handling. BCS supports development of embedded applications for cable and satellite **set-top** boxes, enabling providers to meet the many sophisticated, market-driven needs of the broadcast television customer. This article describes each of the four subsystems that comprise Broadcast Services and provides information about how to implement the subsystems on the **set-top** box. This article is intended for developers (14 printed pages).

Contents

Introduction

Microsoft **Windows CE** is a compact, scalable operating system that enables development of embedded systems, such as for the cable **set-top** box. **Windows CE** uses a subset of the Microsoft Win32 application programming interface (API) commonly used on

Windows-based desktop and server computers. Broadcast Services (BCS) adds components to the **Windows CE** operating system that enable television broadcast suppliers to develop systems to:

- Manage audio/video streams
- Develop electronic program guides
- Provide conditional access (CA) services
- Process broadcast data

BCS assures accessibility to the developer community by using current Microsoft technologies, such as customized Microsoft ActiveX controls and current Microsoft DirectX services. Furthermore, BCS provides broadcast suppliers with a flexible architecture that both supports evolution in **set-top** box hardware design and also protects applications from operating system changes. BCS supports analog, digital, and high-definition television (HDTV) streams, enabling:

- Services acquisition from cable, satellite, and terrestrial sources.
- Analog audio/video input and output, for graphics overlays on current TV signals or for National Television Standards Committee (NTSC) encoders to play video on older TVs.
- Digital TV decoding, such as for Moving Pictures Experts Group (MPEG) 2 decoding and demultiplexing, or for AC-3 audio.

For the interactive home networking environment, BCS will enable support of digital recording and multiple tuners and external devices, such as VCRs for one-step VCR recording from EPGs. One-step recording will be available once the Reminder feature of EPG is fully implemented. BCS is compliant with both U.S. and European industry standards, such as:

- Advanced Television Enhancement Forum (ATVEF), including triggers and crossover links
- Digital Video Broadcasting (DVB), including DVB-C (cable), DVB-T (terrestrial), and DVB-S (satellite)
- Advanced Television Systems Committee (ATSC)
- National Television Standards Committee (NTSC)
- Phase Alternating Line (PAL)
- Sequential Color with Memory (SECAM)

Broadcast Services ActiveX Controls

BCS extends the **Windows CE** operating system through an API of ActiveX controls that enable broadcast suppliers to meet the market-driven needs of television broadcast consumers. You can either write Web applications using DHTML, ECMAScript, or Microsoft JScript development software, or develop native applications using a programming language, such as Microsoft Visual Studio or Microsoft Visual C++. BCS ActiveX controls simplify the process of creating integrated, well-designed BCS applications.

The **Windows CE** BCS API enables television broadcast suppliers to develop applications

that bring interactive television to their customers. BCS supports development of applications that integrate the television with the Internet, and provides access to remote data stores, such as for customer authentication and authorization or for television programming schedules.

You can develop BCS applications to enhance what customers love to do most with the **set-top** boxes – watch TV! Enhanced applications include integrated EPG, parental controls, controlling the VCR, web browsing, e-mail, and so forth. This also includes visual effects, such as combining the broadcast stream with other graphics to make display capabilities easy to use. You can create applications that display on-screen program schedules without forcing the user to change channels. Users can easily view and choose among programming options on dozens of stations available to broadcast customers, without having to suspend the program currently playing. Not only can an application display a translucent electronic program guide over the TV video, it can display the TV video through an embedded window on a Web page that contains other text and graphics.

With the BCS API, broadcast suppliers can also control whether a user receives a channel or a program just as they do with today's **set-top** boxes. However, because of the rich environment and power of BCS, you can create a new generation of pay-per-view (PPV) and impulse PPV, to maximize the business of the premium tiers and paid-for programming. The BCS API is the focus of this article.

What This Article Contains

This article provides information about the **Windows CE** Broadcast Services API of ActiveX controls and how to implement them in applications for the **set-top** box. Such applications may be designed to:

- Present program schedules that overlay the current video image so that the viewer does not have to change channels to read schedules.
- Enable parents to configure the system to be suitable for children, general viewing, or adult-only viewing.
- Restrict access based on available credit for PPV.
- Display ATVEF-enhanced TV content through Web pages.

BCS Architecture Overview

BCS provides a powerful **set** of services by building on the **Windows** infrastructure using ActiveX controls, Microsoft DirectShow™, Microsoft DirectSound, Microsoft DirectDraw, standard **Windows CE** Device drivers, ActiveX Data Objects for **Windows CE** (ADOCE), Network Driver Interface Specification (NDIS), and TCI/IP. BCS adds components to the operating system for A/V stream management, EPG services, and conditional access (see Endnote).

BCS provides four major services:

- *Audio/Video Stream Handling*This subsystem handles tuning to an incoming broadcas stream to receive broadcast audio and video, and allows applications to display

broadcast video and audio mixed with locally generated graphics and audio. Applications do not use the A/V interfaces directly; rather, they use the BCS TVControl.

- *EPG Infrastructure*This infrastructure provides an interface for applications to access an EPG database. It also provides a mechanism for EPG data to be loaded into an EPG database independent of how the EPG data is delivered. Finally, it provides an interface for the rest of the BCS components to access the EPG. The EPG infrastructure clearly separates the delivery of EPG data from application access to the data, which makes it possible to have multiple EPG data providers and multiple EPG user interface providers.
- *Conditional Access* The CA manager deals with purchasing, decryption, and policies related to accessing broadcast streams. BCS supports CA systems that are supplied by various third parties. BCS CA provides a way for vendors to interact with **Windows CE** in a secure manner.
- *Broadcast Data*Broadcast Data Services receives various kinds of broadcast data (including V-chip, Line21, and digital broadcast data streams) and makes them available to applications via Winsock as well as other tailored COM interfaces.

The following illustration provides an overview of the architecture of the BCS subsystems. BCS uses the standard **Windows CE** device driver model. The drivers are written by the Original Equipment Manufacturer (OEM) and are executed by Device.exe. See Figure 1.

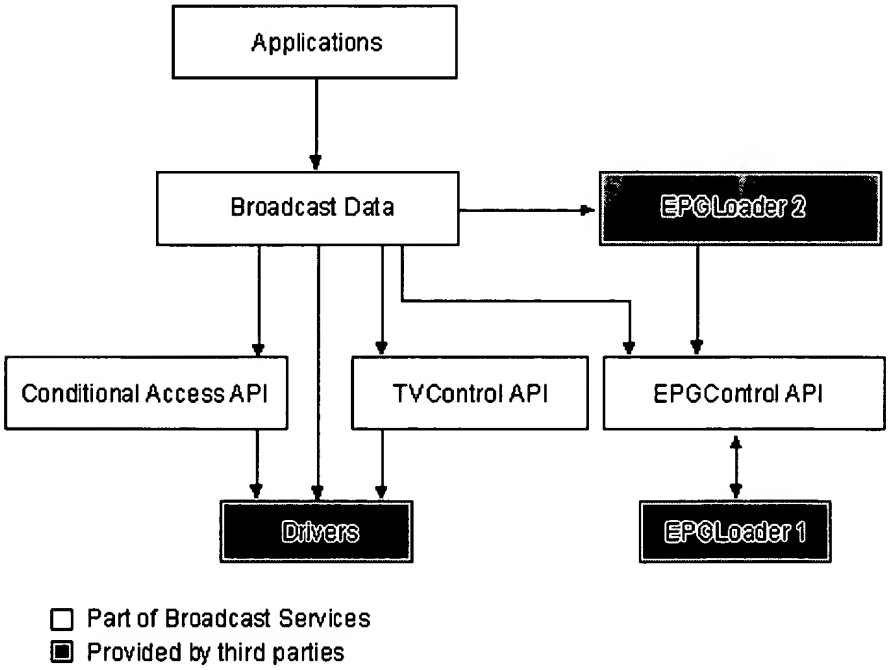

Figure 1. Overview — architecture of BCS subsystems

A/V Manager and DirectX Services

Applications use the TVControl ActiveX control to select broadcast audio and video streams for display. The TVControl uses the A/V manager, which coordinates the BCS subsystems. Applications do not use the A/V manager interfaces directly.

The A/V manager uses a DirectShow filter graph to manage the broadcast streams. Filters expose well-defined interfaces for controlling specific aspects of decoding and displaying broadcast audio and video. Filters provide a mechanism for hardware OEMs to add value while not disturbing the rest of the software architecture. The filters can be used to address the specific hardware, or they can be used to provide software services, such as a software decompressor. OEMs provide the filters and hardware drivers.

Broadcast video is combined with locally generated graphics or Web pages by using DirectDraw. The application may use DirectDraw surfaces.

Implementing DirectShow on the desktop computer differs from **Windows CE**; the desktop computer uses the **Windows** driver model, whereas **Windows CE** uses the Device.exe device drivers. The **Windows** driver model does not exist in **Windows CE**.

Broadcast Data Architecture

The Broadcast Data architecture organizes data in two ways. First, the Broadcast Data facilitates the drivers that receive data by offloading most of the data interpretation chores and providing a simple, low-level interface for delivering data. Second, Broadcast Data also facilitates the applications that use the data by providing a collection of interfaces that simplify broadcast data reception. The application has a choice of three interfaces through which it can access data. It can fetch data directly from Winsock; it can receive data from a COM interface; or it can have the data compiled into a table that it reads directly. Broadcast Data supports three categories of data, which:

- Deliver User Datagram Protocol/Internet Protocol (UDP/IP) datagrams in the traditional way.
- Provide access to the driver's raw, uninterpreted data.
- Provide access to interpreted data. Data is interpreted by deleting duplicates, demultiplexing, aggregating, and parsing, as needed.

V-chip data interpretation provides an example of how Broadcast Data facilitates drivers and applications. The analog TV VBI Line21 driver and the digital TV Picture User Data driver both produce Line21 byte pairs, which they deliver to the Broadcast Data Services without further processing by the driver. Broadcast Data Services demultiplexes and adds these two bytes into an Extended Data Services (EDS) message. When EDS messages are complete, they are sent to NDIS, UDP/IP, and Winsock. If the application is waiting at the Winsock interface, it receives the EDS message. If the application is using the COM interface, it receives a callback with the EDS message. In both cases, the application must then parse the data for the V-chip information. If the application is using the Tabler interface, it receives a callback containing a program information structure from which it can easily read the V-chip information.

Introduction to Conditional Access

The term *conditional access* (CA) refers to restrictions placed on the viewing of TV broadcast content. These restrictions include policy limits relating to ratings or viewing

time, restricted access to subscription-based services, and PPV services.

Windows CE Broadcast Services provides a CA API, which allows applications to receive and respond to CA-related errors and to manipulate purchasable items in a manner independent of the underlying CA technology.

BCS allows CA systems to communicate with both the CA applications and the rest of the BCS system. The CA subsystem supports the following services:

- Decryption of video streams
- Purchase through PPV services
- Enforcement of user policy (such as parental control, V-chip setting, or purchase limits)
- Enabling CA systems to access custom hardware, software, and smart card capabilities
- Implementation of multiple CA systems

Applications initiate usage of CA services through the TVControl ActiveX control. TVControl allows control of the user interface for customer interactions required by the CA system.

EPG

The EPG services collect transient and remote guide listings data and store it to provide fast, rich, read-only querying capabilities. The application developer accesses the guide listings data through the EPGControl, a non-visual ActiveX control that gives applications easy access to the EPG data, without having to know the underlying database schema. The EPG infrastructure consists of five main components:

- The EPG ActiveX control provides applications with a simple interface for accessing the EPG database.
- The EPG Loaders and Writers provide facilities for loading the EPG database with the EPG data from a variety of EPG data providers over any number of data transport media (VBI, DOCSIS, or MPEG sections).
- The EPG Data Map provides a simple **set** of interfaces that are required by the rest of the BCS system.
- The EPG database contains the tables that contain the EPG data.
- ADOCE provides an API for accessing the EPG database.

The EPG infrastructure provides a standard way to access EPG data from BCS. The EPG database can be built using the **Windows CE** file system or any database that supports ADO.

The EPG services themselves do not provide a graphical representation of the data. The application developer must implement the graphical interface.

The system integrator provides one or more EPGLoaders to collect guide-listings data from in-band or out-of-band television signals, from HTTP or FTP, from TCP/IP sockets, or from any other communications protocol. The EPGLoaders use the EPGControl to read data

stored by EPG services and compare it to the incoming data. If necessary, the EPGLoaders add new data to the EPG services store through the EPGWriter. EPG services enable the following functionality:

- Scalability of Data
- Extensibility
- Collecting and Storing Data
- Retrieving Data

Instantiating Controls in HTML Pages

TVControl

The TVControl scripting interface makes it easy to use Broadcast Services from an HTML environment. Typically, you declare your TVControl object as shown in the following JScript sample code. The CLSID must be as shown:

```
<OBJECT CLASSID="CLSID:C82FB020-62D1-11D2-98DB-0060083176E3" ID=TVControl>
<SCRIPT LANGUAGE=JScript>
TVControl.TuneChannel(0, 20);    // Tunes to Channel# 20
</SCRIPT>
```

The TVControl is a windowless control. This means that it needs to be used from an application or a browser that hosts the control and then displays the video in the region of the window of that application or browser. The advantage of this windowless control is tha the control can display video in any region, as programmed by the application or browser. Fewer resources being consumed and faster activation/deactivation lead to performance advantages.

For this version, only one instance of a TVControl is permitted. In the future, multiple instances of the TVControl will be supported.

EPGControl

The EPGControl scripting interface makes it easy to use EPG services from an HTML environment. Typically, the EPGControl object is declared as in the following JScript sample code:

```
<OBJECT CLASSID="CLSID:C653DFC8-9884-11D2-9299-00A0C9C9E689" ID=EPGControl>
<SCRIPT LANGUAGE=JScript>
if (EPGControl.IsAnyDataAvailable) // Check EPG store for data.
{
   // Continue.
}
</SCRIPT>
```

Application Programming Interfaces

The next sections present an overview of the BCS ActiveX control APIs.

TVControl

TVControl is an ActiveX control that provides the application interface to the A/V manager. Applications can be written in a suitable language, such as JavaScript, ECMAScript, JScript or Visual C++. The A/V manager runs in the background and has access to all the devices. A user controls the A/V manager through the TVControl.

When using the TVControl, the user can change channels with a remote control, as with a traditional TV. In the interactive world, there are new possibilities for changing channels. For example, channels can be changed by going to a Web page for a particular network. The page might use a TVControl and provide a window within the Web page tuned to the local affiliate. Figure 2 illustrates what the TV screen might look like.

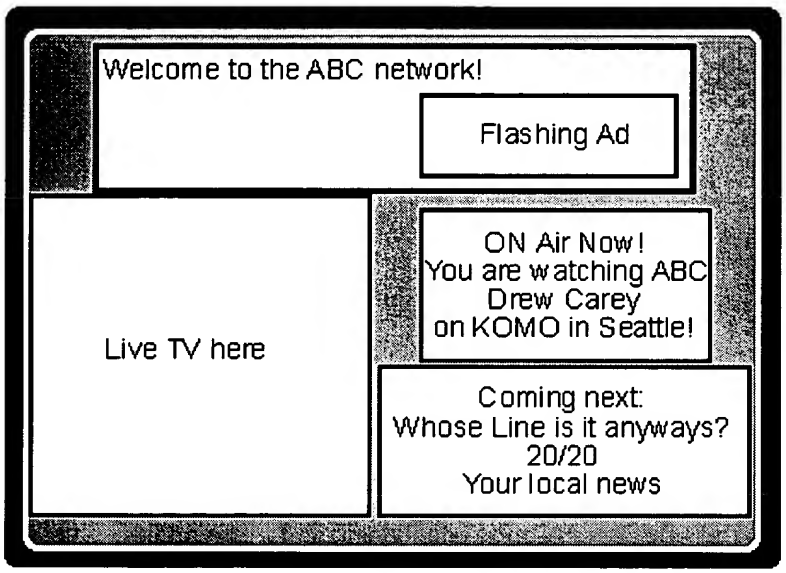

Figure 2. Tuning criteria using TVControl

The TVControl can tune by channel, station, network, source identifier, and selector identifier. For additional tuning criteria, you should use the EPG control to map to TVControl properties. TVControl provides other features in addition to changing channels. Applications can use the TVControl to:

- Enable alternate audio and video tracks
- Display a UI of thumbnail images of recently viewed channels
- Customize the dimensions of the displayed video
- Change aspect ratios on input and output
- Control the audio to mute or **set** the volume

EPG Controls

The EPG controls are ActiveX controls that provide applications with easy access to the guide data without requiring them to directly access an underlying database schema or raw data stream. The EPG services collect transient and remote guide listings data and store them to provide fast, rich, read-only querying capabilities.

The EPG database is automatically constructed by the EPG services to update the guide database. The EPG will be a primary application built for a **set-top** box. It can access all broadcast stream-handling services using the TVControl. Figure 3 provides an example EPG with live video embedded in the upper-right corner of the guide.

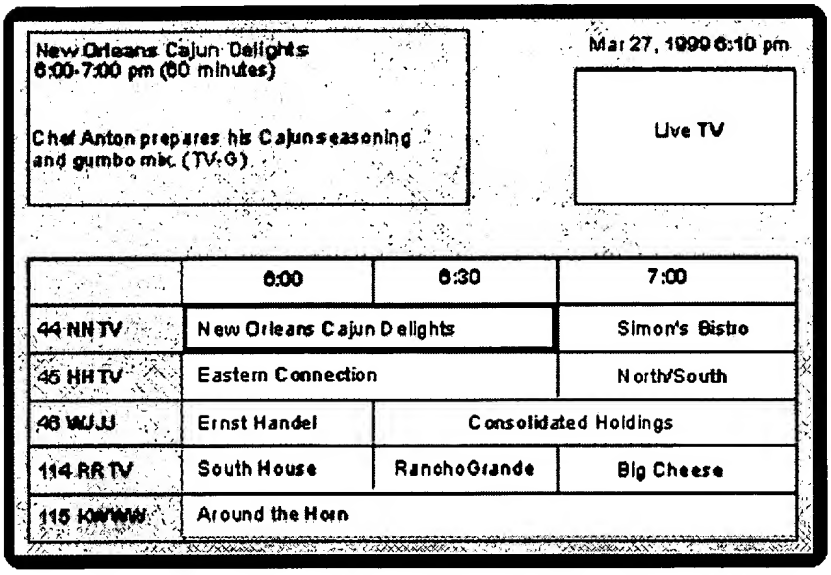

Figure 3. EPG with embedded live video

You can access the guide listings data through the EPGControl, a nonvisual ActiveX control that provides abstracted access to the data store. The EPG services themselves do not provide a graphical representation of the data, as shown in the above illustration. You must implement the graphical interface.

EPG Architecture Overview

The following illustration in Figure 4 shows an overview of the EPG services architecture.

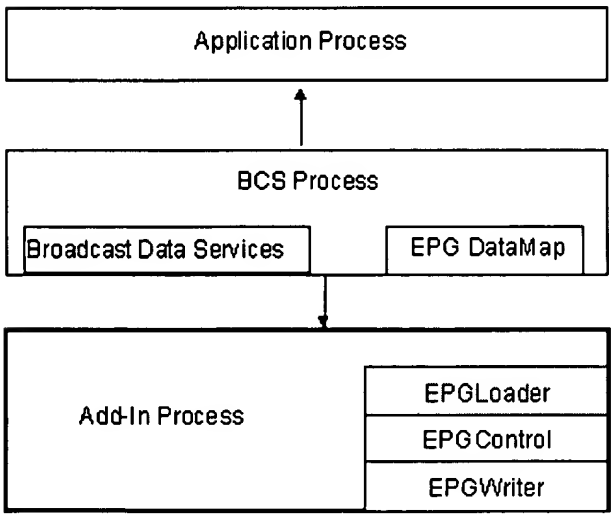

Figure 4. Overview — EPG services architecture

The system integrator provides one or more EPGLoaders to collect guide listings data from in-band or out-of-band television signals, from HTTP or FTP, from TCP/IP sockets, or from any other communications protocol. EPGLoaders use the EPGControl to read data stored by EPG services and compare it to the incoming data. If necessary, the EPGLoaders add new data to the EPG services store through the EPGWriter.

A description of the EPG ActiveX controls follows.

EPGControl—Performs queries and retrieves information about channels and schedules.

EPGWriter—Stores collected guide listings data in the EPG services data store so that the data can be queried and retrieved by the EPGControl.

EPGDataMap—Obtains necessary data for the A/V manager from the EPG services. This interface should not be used by EPGLoaders or other applications or components developed by system integrators or application developers.

Scalability of Data

Although the device capabilities will determine the amount of local memory or disk storage available for guide listings data, with EPG controls you can modify the amount of data stored on any of the following axes: time, channels, language, or richness.

For example, for richness you can modify the amount of data stored for titles, descriptions and other program description attributes. Figure 5 provides an example of richness scaling.

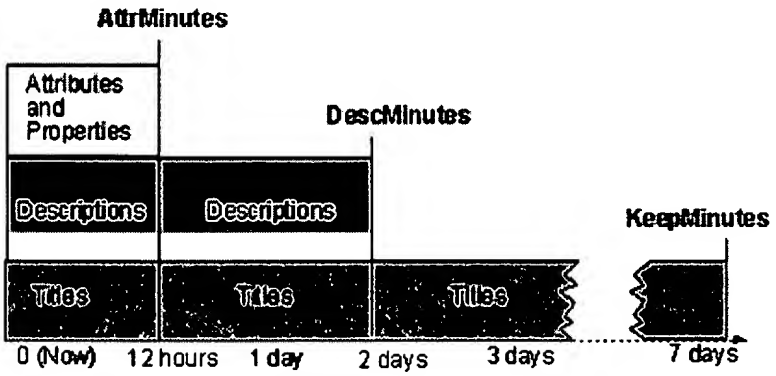

Figure 5. Example of richness scaling

The richness of program information can vary according to title, description, and other attributes (for example, closed-captioning). In addition, if a channel is designated as a favorite channel, the richness of programs on this channel can be specified independently.

Extensibility

The EPG services architecture is flexible and allows for new sources of data and new data sets.

As new sources of data become available, you can add additional EPGLoaders. For example, a new EPGLoader can be written to collect movie review data from an HTTP address.

In addition, as data sources add new properties to guide listings data, EPGLoaders can be modified to add this new data to the EPG services store. A modified graphical interface application can then retrieve the data.

Extensible properties can be added to the channel, program, schedule entry, or Web link data. Multiple properties can be added to the same data **set**.

New properties are stored and retrieved using a flexible name-value scheme. Each new attribute is named, and the value is stored along with that name. Values are stored and retrieved using a textual string representation. Applications can convert this string value back to its native format using the C/C++ VARIANT class or through JScript constructors that take a string.

For example, an EPGLoader can add a new program property named **RogersReview** to al programs with the values THUMBSUP or THUMBSDOWN. The graphical application can then use the EPGControl to request the value for **RogersReview** for any program. Because the extensibility mechanism supports multiple properties, a second property can also be added named **FidosReview**, with values such as PAWSUP or PAWSDOWN.

EPG Controls provide other features in addition to scalability of data and extensibility.

Other Features

Applications can use the EPG controls for:

- Data Collection
- Data Storage and Validation
- Maximum Storage Size Regulation
- Element Storage Size Regulation
- Storage Update and Removal
- Multiple Loader Support
- Data Collisions
- Time-Based Loading
- Reminders
- Record Events

CA Controls

The CA controls are ActiveX controls that enable applications to monitor conditional access privileges. CA controls allow applications to receive and respond to CA-related errors and to manipulate purchasable items independent of the underlying CA technology.

BCS also provides a COM-based interface, which allows CA systems to communicate with the CA applications and the rest of the BCS system. The CA subsystem supports the

following services:

- Decryption of video streams
- Purchase through PPV services
- Enforcement of user policy (such as parental control, V-chip setting, or purchase limits)
- Enabling of providers to access custom hardware, software, and smart card capabilities
- Implementation of multiple CA service providers

Applications initiate usage of CA services through the TVControl ActiveX control. TVControl allows control of the user interface for customer interactions required by the CA system. The OEM must provide a CA service provider.

With BCS, conditional access providers can:

- Eliminate the need for applications to change if a CA system is added, changed, or replaced
- Allow multiple CA systems to coexist in the same box
- Ensure that CA systems have independence to define the business model and the business offerings, policies, and other business details
- Allow providers flexibility for their own specific hardware and software requirements and capabilities

For example, purchasing applications can contain the business rules, whereas the CA interfaces transmit only necessary data. This enables broadcast suppliers to respond quickly to market opportunities. They can change the business rules without changing the **Windows CE** operating system and without changing the CA system software.

The BCS CA subsystem enables broadcast suppliers to develop new classes of premium content applications and purchasing applications while independently evolving the underlying CA technology. This allows for continuous improvement and enhancement in performance and features offered.

CA Subsystem Architecture Overview

The CA subsystem in BCS provides an abstraction layer between entitlements enforced by hardware installed on a host **set-top** box, a limits policy, and the application.

Figure 6 provides an overview of the CA subsystem architecture.

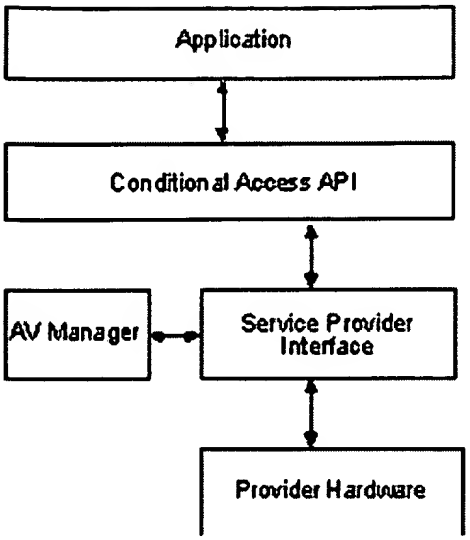

Figure 6. Overview — CA subsystem architecture

The application discovers and manipulates CA-related information via a **set** of four ActiveX controls with common interfaces. By exposing CA functionality in a generic fashion, applications can separate themselves from the details of the CA hardware and limits policies. Furthermore, CA suppliers can upgrade their CA systems, independent of the applications. A description of the CA ActiveX controls follows.

CAErrorSource—Fires ActiveX events into the application when an access denial occurs. This control allows the application to discover and process denial events regardless of whether the TVControl is hosted by the application directly or inside a Web page hosted by an application.

CAOfferItem—Represents a purchasable item. It can be instantiated from a Web page through a string of data. Purchasable items can also become apparent to the application through a CAError or the CAOfferList control.

CAOfferList—Defines a simple, uniform way to access and search through pending offers and recently purchased items.

CAProviders—Allows the application to enumerate and query the provider software modules currently installed in the **set-top** box.

Service Provider Interface

Provider modules are DLLs that are enumerated in the system registry and loaded by the CA manager at system start time. The interface between the provider modules and the CA manager is called the service provider interface (SPI). The SPI enables:

- Multiple entitlement systems to be present and active simultaneously.
- Implementation of a sophisticated limits policy.

To integrate a CA system into BCS, the CA provider must author a provider software module. Additionally, to implement policy limits, a special provider software modulethe

Limits Providermust be developed. Although many CA provider software modules may be active in the system at once, only a single Limits Provider is allowed.

Provider modules can be identified by a ProviderID globally unique identifier (GUID), which is listed in the registry with the rest of the provider module configuration information. ProviderIDs are used in OfferStrings, which are external representations of purchasable items. ProviderIDs are exposed to the application in error information and are exposed by means of the CAProviders control.

Conclusion

Based on trends in the industry and market-driven needs, television broadcast consumers expect greater integration of their televisions with the Internet. BCS extends the **Windows CE** operating system so that broadcast suppliers can develop interactive applications that meet the needs of their customers. With BCS ActiveX control APIs, you can easily design and build applications to run on **set-top** boxes, offering customers functionality such as electronic program guides, conditional access monitoring, and targeted programming based on viewer profiles.

With the BCS APIs, you can create programs to meet these needs and look toward the future needs of the television broadcast consumer. The BCS architecture is flexible enough to support proprietary systems where vendors can offer additional features, control functions, and services that create a very compelling, integrated experience for the home television viewer.

For More Information

For more information about **Windows CE**, see the Microsoft **Windows CE** Web site at: www.microsoft.com/**windows**/embedded/default.asp.

For more information about Broadcast Services for **Windows CE**, see the **Windows CE Set-Top** Box Guide and Reference supplied with the **Set-Top** Box Kit.

Endnote

Conditional Access (CA) refers to restrictions placed on the viewing of TV broadcast content. These restrictions include policy limits relating to ratings or viewing time, restricted access to subscription-based services, and pay-per-view (PPV) services.

Microsoft Windows CE .NET 4.2

Block Driver Architecture

Drivers for **block** devices generally expose the stream interface, and the **block** devices appear as ordinary disk drives. Applications access files on a **block** device through standard file APIs such as CreateFile and ReadFile.

The application calls the **ReadFile** function using a handle to a file that is stored on the **block** device. The FAT file system, translates the read request to logical blocks. The FAT file system searches the buffer cache for the requested blocks. If these are not present, it issues an IOControl request to the corresponding **block** device **driver** to read bytes from the **block** device. Then, the **block** device **driver** receives the **IOControl** request, and then fulfills the request by accessing the **block** device through one of its low-level interfaces.

See Also

Block Drivers | File Systems | **Block Driver** Samples | **Block Driver** Registry Settings | **Block Driver** Manager | **Block** Device File Systems | File System Loading and Unloading | **Block Driver** Interface | **Block Driver** Loading | **Block Driver** Installation | **Block Driver** Detection | **Block Driver** Access | **Block Driver** Power Cycle

---

Last updated on Tuesday, April 22, 2003

Microsoft Windows CE .NET 4.2

File Systems

Windows CE offers three types of file systems:

- A ROM-based file system
- A RAM-based file system
- A FAT file system for Advanced Technology Attachment (ATA) devices, flash memory, and SRAM cards

In addition, embedded systems developers can create and register proprietary file systems.

Regardless of the type of storage, all of the file systems are accessed through the Microsoft Win32® file-system application programming interface (API).

As with the Microsoft Windows-based desktop platforms, Windows CE uses handles for file access. The CreateFile function returns a handle that references the created or opened file. Subsequent read, write, and information functions all use that handle to determine the file on which to act. File read and write functions also use a file pointer to specify the location within a file in which the read or write operation takes place.

Windows CE uses a variety of techniques to simplify memory management and to reduce memory overhead. First, Windows CE does not use the current directory concept. Instead, all of the references to an object are given in the full path. Also, Windows CE automatically compresses all of the files in the object store, based on an OEM option; therefore, no file has a flag to indicate compression. Of course, a flag does exist that distinguishes between a file in ROM and a file in RAM.

A file can be up to 4 GB. In addition to the object store, a user can install a file system such as the FAT file system. An installed file system can provide access to a PC Card or to other external storage devices. You can divide an external storage device into multiple volumes, each of which is mounted separately. Each mounted volume is visible to the user as a folder in the root directory of the installed file system. While you can back up data to an external storage device, the working registry and RAM file system can exist only in the object store.

Note For programming purposes, Windows CE considers the object store to be a special type of volume that is always mounted.

See Also

Object Store and Registry | Storage Manager | Partition Manager | Partition Driver | Block Drivers

---

Last updated on Tuesday, April 22, 2003

Microsoft Windows CE .NET 4.2

Block Driver Samples

The following list shows the sample block device drivers:

- Sample ATADISK Driver
- Sample ATAPI Driver
- Sample Secure Digital Card Driver

See Also

Block Drivers | Block Driver Architecture | Block Driver Registry Settings

---

Last updated on Tuesday, April 22, 2003

Microsoft Windows CE .NET 4.2

Sample ATADISK Driver

The sample ATADISK driver is in the %_WINCEROOT%\Public\Common\OAK\Drivers\Block\ATADisk directory. In addition to the standard stream interface functions, the ATADISK driver also exports a PC Card Plug and Play detection function, **DetectATADisk**. The detection function only reads the attribute space of the PC Card; it does not read any data or perform any write operations on the PC Card. The function looks for disk device type 4 in the PC Card's **CISTPL_FUNCID** tuple, and for ATA device type 1 in the type 1 **CISTPL_FUNCE** tuple.

The presence of the **HKEY_LOCAL_MACHINE\Drivers\PCMCIA\ATADisk** registry key causes the Device Manager to call the driver's **DetectATADisk** function when you insert a PC Card and there is no driver that is associated with the card's Plug and Play identifier. For more information about the Device Manager, see Device Manager. The following registry key example shows this:

```
[HKEY_LOCAL_MACHINE\Drivers\PCMCIA\Detect\50]
    "Dll"="ATADISK.DLL"
    "Entry"="DetectATADisk"
```

When **DetectATADisk** detects an ATA-compatible PC Card, it causes the Device Manager to load the driver listed in the **HKEY_LOCAL_MACHINE\Drivers\PCMCIA\ATADisk** registry key. The following registry key example shows this:

```
[HKEY_LOCAL_MACHINE\Drivers\PCMCIA\ATADisk]
    "Dll"="ATADISK.DLL"
    "Prefix"="DSK"
    "Ioctl"=dword:4
    "Profile"="PCMCIA"
    "IClass"=multi_sz:"{A32942B7-920C-486b-B0E6-92A702A99B35}","{A4E7EDDA-E575-4252-9D6B-4195D48BB865}"
```

The following table shows optional values that can go under **HKEY_LOCAL_MACHINE\Drivers\PCMCIA\ATADisk**; these values affect all devices that the ATADISK driver effects.

Key	Description
Cylinders	A value of ***xxx*** causes the ATADISK driver to not rely on the number of cylinders that the ATA device reports in response to the ATA IDENTIFY command. The number specified by the **Cylinders** registry value is used.
Heads	A value of ***hh*** causes the ATADISK driver to not rely on the number of heads that the ATA device reports in response to the ATA IDENTIFY command. The number specified by the **Heads** registry value is used.
Sectors	A value of ***ss*** causes the ATADISK driver to not rely on the number of sectors per track reported by the ATA device in response to the ATA IDENTIFY command. The number specified by the **Sectors** registry value is used.
CHSMode	A value of 1 forces the ATADISK driver to use Cylinder/Head/Sector (CHS) addressing mode. If the **CHSMode** value is 0 or the value is not present, the ATADISK driver uses the addressing mode reported by the ATA device in response to the ATA IDENTIFY command. The ATADISK driver uses logical block address (LBA) mode when available.

See Also

Block Driver Architecture | Block Driver Registry Settings | PC Card Drivers

Last updated on Tuesday, April 22, 2003

Microsoft Windows CE .NET 4.2

Sample ATAPI Driver

The sample ATAPI driver is in the %_WINCEROOT%\Public\Common\OAK\Drivers\Block\ATAPI directory. The following table shows the ATAPI driver IOCTLs.

IOCTL	Description
IOCTL_CDROM_DISC_INFO	Retrieves disc information to fill in the CDROM_DISCINFO structure.
IOCTL_CDROM_EJECT_MEDIA	Ejects the CD-ROM.
IOCTL_CDROM_GET_SENSE_DATA	Retrieves CD-ROM sense information and fills in the CD_SENSE_DATA structure.
IOCTL_CDROM_ISSUE_INQUIRY	Retrieves information used in the INQUIRY_DATA structure.
IOCTL_CDROM_PAUSE_AUDIO	Suspends audio play.
IOCTL_CDROM_PLAY_AUDIO_MSF	Plays audio from the specified range of the media.
IOCTL_CDROM_READ_SG	Reads scatter buffers from the CD-ROM and the information is stored in the CDROM_READ structure.
IOCTL_CDROM_READ_TOC	Returns the table of contents of the media.
IOCTL_CDROM_RESUME_AUDIO	Resumes a suspended audio operation.
IOCTL_CDROM_STOP_AUDIO	Ends audio play.
IOCTL_CDROM_TEST_UNIT_READY	Retrieves disc-ready information and fills the CDROM_TESTUNITREADY structure.

See Also

Block Driver Architecture | Block Driver Registry Settings

Last updated on Tuesday, April 22, 2003

Microsoft Windows CE .NET 4.2

Sample Secure Digital Card Driver

The sample Secure Digital card driver supports both MultiMedia Card (MMC) and Secure Digital (SD) cards and is designed for a PCI-based card developed by SanDisk, part # SDSDEV-01. The MMC supports unsecured storage. The SD card has both a nonsecure and a secure area. The nonsecure area is 90 percent of the space; the remaining 10 percent is secure. This driver only supports the nonsecure portion of the SD card.

Polling the socket approximately once per second detect insertions and removals. When the state changes, the socket driver notifies and updates the file system.

The sample Secure Digital card driver exports a stream interface with the DSK prefix and the block device driver interface. For more information about the stream interface, see Stream Interface Drivers.

See Also

Block Driver Samples | Block Driver Architecture | Block Driver Registry Settings

---

Last updated on Tuesday, April 22, 2003

Microsoft Windows CE .NET 4.2

Block Driver Manager

When the system loads a block driver, the system informs the Storage Manager of the load event. The Storage Manager consists of the Block Driver Manager, the Partition Manager, and the File System Driver (FSD) Manager. For more information, see Storage Manager. When a block driver loads, the Storage Manager queries the block driver and the registry and informs the Partition Manager, which is responsible for managing logical partitions on a storage device, to load the device. You can use the storage APIs to enumerate and manage various block storage devices that the system mounts. You cannot enumerate storage devices that do not have a block driver.

See Also

Block Driver Architecture | Storage Manager | Partition Manager | Block Driver Samples | Block Driver Registry Settings | Block Device File Systems | File System Loading and Unloading | Block Driver Interface | Block Driver Loading | Block Driver Installation | Block Driver Detection | Block Driver Access | Block Driver Power Cycle

---

Last updated on Tuesday, April 22, 2003